



#10
30/3

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: James McKeeth

§ Group Art Unit: 2122

Serial No.: 09/449,782

§

§

Filed: November 26, 1999

§

§

Examiner: Mary J. Steelman

For: Command Line Output
Redirection

§

§

Atty. Dkt. No.: MCT.0132US
99.03185

Board of Patent Appeals & Interferences
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, Virginia 22313-1450

RECEIVED

JUL 02 2003

APPEAL BRIEF

Technology Center 2100

Dear Sir:

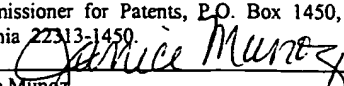
Applicant hereby appeals from the Final Rejection dated January 24, 2003, finally
rejecting claims 1-22.

I. REAL PARTY IN INTEREST

The real party in interest is Micron Technology, Inc.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

Date of Deposit:	June 24, 2003
I hereby certify under 37 CFR 1.8(a) that this correspondence is being deposited with the United States Postal Service as first class mail with sufficient postage on the date indicated above and is addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.	
	
Janice Munoz	

III. STATUS OF THE CLAIMS

Claims 1-22 have been finally rejected and are the subject of this appeal.

IV. STATUS OF AMENDMENTS

The claims have not been amended subsequent to the final Office Action of January 24, 2003.

V. SUMMARY OF THE INVENTION

Referring to FIG. 2 of the specification of the present application, redirection routine **200** in accordance with one embodiment of the invention uses a user/application specified identifier (block **202**) to identify command line utility output (block **204**) which it stores in a system-wide storage location (block **206**). By system-wide, it is meant that the storage location is available to all user applications and is, furthermore, maintained by operation of the underlying operating system. Following the act of storage in block **206**, a value associated with the identifier in the system storage is updated to indicate completion of the redirection routine and to, possibly, provide additional information to the calling application such as the amount (e.g., number of lines) of information stored. Once redirection routine **200** completes the act of storing in block **206**, the application invoking routine **200** may use the specified identifier to access the stored command line utility output. Specification, p. 3.

One benefit of a redirection routine in accordance with FIG. 2 is that the calling application does not need file creation authority -- no temporary files are created. Another

benefit is that there is no need for the calling application to remove temporary files as in prior art techniques such as that illustrated in FIG. 1. A corollary of this benefit is that the calling application does not require file deletion authorization. Yet another benefit of a redirection routine in accordance with the invention is that a second application cannot inadvertently destroy the results generated by a first application by accidentally replacing or deleting a temporary file (e.g., a background process designed to remove temporary files). Still another benefit of the invention is that the application invoking redirection routine 200 does not have to have disk I/O (input-output) authority as the storage location is maintained by the underlying operating system -- the application makes I/O calls to the specified storage location through standard system calls (see discussion below).
Specification, pp. 3-4.

By way of example, consider a situation in which an executing application needs information of the type provided by command line utility CMD-UTIL, where CMD-UTIL represents any utility executable from a command line prompt (e.g., the "dir" directory command of a Microsoft WINDOWS[®] operating system or the "head" command of a UNIX[®] operating system). In accordance with the invention, the application invokes a system call of the form:

CMD-UTIL [PARAM] | REDIRECT ID

Here, [PARAM] represents zero or more parameters that control or modify the execution of the CMD-UTIL utility, the "|" symbol represents the piping function available in many

operating systems such as WINDOWS[®], UNIX[®] and derivatives thereof, REDIRECT is the name of routine 200, and ID is one or more parameters which REDIRECT routine 200 associates with output from CMD-UTIL during the act of storage in block 206 of FIG. 2. Specification, p. 4.

It will be recognized that the calling application will generally ensure that the identifier it passes to routine 200 has either not been used or may be reused. It will further be recognized that command utilities may be stacked. That is, output from a first command utility (CMD-UTIL-1, for example) may be piped to a second, third, or Nth command utility (CMD-UTIL-N, for example) which may then be piped to routine 200. In this case, a system call in accordance with the invention would be:

CMD-UTIL-1 [PARAM] | . . . | CMD-UTIL-N [PARAM] | REDIRECT ID,

where ". . ." represent one or more commands of the form CMD-UTIL-X [PARAM].

Specification, pp. 4-5.

Because many current personal computer systems (PCs) are operated or controlled by one version or another of the Microsoft WINDOWS[®] operating system, an illustrative embodiment of redirection routine 200 utilizing the WINDOWS[®] system registry (hereinafter, the registry) will now be given. It will be recognized that the registry is an operating system generated and maintained database which application programs, application setup programs, and the operating system itself use to store configuration information. Specification, p. 5.

Information stored in the registry is organized into hierarchical keys and associated key entries. Current versions of the registry use six predefined root keys (AKA Hives): HKEY__USERS; HKEY.CLASSES.ROOT; HKEY.CURRENT.USER; HKEY.CURRENT.CONFIG; HKEY_LOCAL.MACHINE; and HKEY.DYN.DATA. Each key in the registry can have one or more sub-key entries. Each key and sub-key can have one or more names (a unique character string identifier) and each name can have an associated value (data stored in a defined manor, may be a character string, binary data, a number, a Boolean value, etc.). Each key and sub-key has one default key entry that has no name. Specification, p. 5.

Access to the registry is provided through system calls defined in the registry application programming interface (API). Illustrative registry API functions include: RegEnumKeyEx, which enumerates the sub-keys of a specified key; RegOpenKeyEx, which opens and returns a handle to a specified key; RegEnumValue, which enumerates the key entries associated with a specified key; RegQueryValueEx, which returns the assigned value of a specified key entry; RegSetValueEx, which assigns a value to a specified key entry, creating the key entry if the key entry was not previously registered; RegDeleteKey, which removes a key from the registry; and RegDeleteValue, which removes a key entry from the registry. Using keys (hereinafter understood to include sub-keys) and registry API system calls, routine 200 can store command line utility output in the registry file. Using the same keys, an application program can retrieve information previously stored by routine 200. Specification, pp. 5-6.

Referring now to FIG. 3, in one embodiment WINDOWS[®] based redirection routine 300 receives an identifier comprising a key from a calling application (block 302). An illustrative key is HKEY.DYN.DATA/CMD-UTIL-OUTPUT-KEY. Routine 300 then begins receiving output from the CMD-UTIL utility, generally one line at a time as most command line utilities generate output targeted for line oriented standard output devices such as a computer display (block 304). The received line is stored in the registry at a key name that uniquely identifies the line (block 306). For example, each received line of output may be stored in the registry key:

HKEY. DYN. DATA/CMD-UTIL-OUTPUT-KEY,

with a name of "N," where "N" is set equal to 1 for the first received line, 2 for the second received line, and so forth. A test is then made to determine if additional command line utility output is available for storage (diamond 308). If another line of output is available (the "yes" prong of diamond 308), processing continues as block 304. If no more output is available (the "no" prong of diamond 308), the default value of the received key (i.e., HKEY_DYN_DATA/CMD-UTIL-OUTPUT-KEY) is set equal to a value corresponding to the total number of lines received and stored by routine 300 (block 310). On completion, output from the command line utility CMD-UTIL is available for retrieval and manipulation by the calling application without the need to create, maintain or delete a temporary file. Specification, p. 6.

In another embodiment, the ID parameter includes a storage location identifier. One value of the storage location identifier may direct use of the registry (or a similar operating system maintained database) while another value of the storage location identifier may direct use of operating system shared memory (e.g., volatile random access memory). One example of operating system shared memory is the "clipboard" memory maintained by the WINDOWS® operating system. Specification, pp. 6-7.

Referring now to FIG. 4, illustrative computer system 400 in accordance with one embodiment of the invention includes redirection routine 400 (e.g., a routine in accordance with 200 and/or 300) to redirect output from a command line utility to a specified operating system controlled memory location. As shown, routine 400 may be retained in storage device 404 which is coupled to processor 406 via system bus 408. It will be understood that storage device 404 may represent non-volatile memory devices or a combination of volatile and non-volatile memory devices. Illustrative non-volatile storage devices include, but not limited to: semiconductor memory devices such as EPROM, EEPROM, and flash devices; magnetic disks (fixed, floppy, and removable); other magnetic media such as tape; and optical media such as CD-ROM disks. It will be further recognized that computer system 400 may incorporate one or more input-output devices 410 such as one or more secondary bus bridge circuits, memory controllers, accelerated graphics port devices and network interface adapters. Specification, p. 7.

Other embodiments are within the scope of the claims on appeal.

VI. ISSUES

- A. **Can a reference that does not disclose the elements of claims 1, 15 and 21 anticipate claims 1, 6-11, 15-16 and 17-22?**
- B. **Can a reference that does not disclose the elements of claim 2 anticipate claims 2-5?**
- C. **Can a reference that does not disclose the elements of claim 12 anticipate claims 12-14?**

VII. GROUPING OF THE CLAIMS

Claims 1, 6-11, 15-16 and 17-22 can be grouped together; claims 2-5 can be grouped together; and claims 12-14 can be grouped together. The claims of each group stand and fall together.

VIII. ARGUMENT

All claims should be allowed over the cited references for the reasons set forth below.

- A. **Can a reference that does not disclose the elements of claims 1, 15 and 21 anticipate claims 1, 6-11, 15-16 and 17-22?**

Claims 1, 15, and 21 recite receiving an identifier and receiving output from a command line utility. The command line utility output is stored in system storage at a location identified by the identifier.

Independent claims 1, 15, and 21 stand rejected under 35 USC § 102 as being anticipated by Buxton (U.S. Patent No. 6,182,279). Buxton does not disclose receiving output from a command line utility and storing the command line utility output in a

system storage. Buxton, in Figure 2, shows the elements comprising a component customization and distribution system that provides a template builder utility (204 and Figure 4A) which enables a base component 202 to be selectively modified and the modifications to the base component stored as a template. The templates are stored in a template storage file 212 with the assistance of template storage dynamic link library (DLL) 205. As shown in Figure 4B, each template 420 contains initialization data 425 representing the modifications to data of the base component, and one or more user-defined instructions 445 useful in utilizing the modifications or customizations to the base component.

In Buxton, component system 200 shown in Figure 2 is a standalone application or is used with Lotus Notes or any software application to implement Object Linking and Embedding (OLE) controls. Component system 200 includes software to Chart, Comment, Draw/Diagram, File View, and Project Schedule, in addition to a template builder utility and a component loader utility. Buxton in column 8, lines 45-52 describes a user interface that enables a user to interact with component system 200 and may be implemented with a simple command line interpreter or may have a more sophisticated graphic user interface with pull down menus to select various options available, such as selection of a specific component, component loader 206, template builder 204, and so forth. However, Buxton does not disclose receiving output from a command line utility and storing that output in system storage at a location identified by a received identifier. All Buxton discloses is using a command line interpreter or graphic user interface with pull down menus to select various available options.

Within Buxton's component system 200, a separate template builder application 204 with its own graphical user interface (GUI) is present as described in column 13. The template builder application allows a user through the GUI to select a base component, customize the component, and store the customizations as a template. The GUI enables users to perform a variety of different actions (i.e. New Template, Open Template, Save, Create Distribution Pack, and Exit Template Builder). After the user has specified the customizations within the component using the GUI and an editor and saved the customizations as a template using the template builder utility, the template is stored in a template storage file with the assistance of a template storage dynamic link library. Buxton does not disclose receiving output from a command line utility but rather teaches that the user through a GUI and editor inputs customizations to the base component. The customizations input by the user are saved as a template and do not generate outputs like the applicant's command line utility, the output then stored in a system storage. In Buxton, the user's inputs are stored as a template in a template storage file. Storage of a template in a template storage file is not storage of command line utility output in a system storage.

Column 13, lines 8-14 describes creation of templates using DLLs that include formatting and storage retrieval functions and procedures to manage and simplify the storage and registration of templated components for the template builder, component loader, and template installer. Thus, column 8, lines 45-52, column 13, lines 8-14 or any other part of the Buxton reference does not disclose receiving output from a command line utility and storing the command line utility output in a system storage.

Therefore, the rejections of claims 1, 5-11, 15-16 and 17-22 should be reversed.

B. Can a reference that does not disclose the elements of claim 2 anticipate claims 2-4?

Claim 2 depends from claim 1 and adds that the act of receiving an identifier comprises receiving an identifier that identifies one or more entries in a system registry database.

Claim 2 stands rejected under 35 USC 102 as anticipated by Buxton. Buxton does not disclose receiving an identifier that identifies one or more entries in a system registry database. Buxton as mentioned above in Figure 2 shows that templates are stored in a template storage file 212 with the assistance of template storage dynamic link library (DLL) 205. Template storage DLL 205 performs a number of formatting and storage methods which manage the storage and registration of templated components. Template storage DLL 205 ensures all additional registry keys and library types are created for template 420 so that spoofing of a templated component makes the templated component appear to be the base control of the original component. Thus, Buxton does not disclose receiving an identifier that identifies one or more entries in a system registry database but rather creation of registry keys for spoofing of templated components.

Thus, the rejection of claims 2-4 should be reversed.

C. Can a reference that does not disclose the elements of claim 12 anticipate claims 12-14?

Claim 12 depends from claim 1 and adds that the act of storing comprises associating each line of command line utility output with a line identifier in the system storage.

Claim 12 stands rejected under 35 USC 102 as anticipated by Buxton. Buxton does not disclose that the act of storing comprises associating each line of command line utility output with a line identifier in the system storage. Buxton, as mentioned above, in Figure 2 shows the elements of a component customization and distribution system that provides a template builder utility (204 and Figure 4A) which enables a base component 202 to be selectively modified and the modifications to the base component stored as a template. Buxton does not disclose storing in a system storage each line of command line utility output with a line identifier but rather in col. 13, lines 35-44 that each template is stored in an ISTORE whole name is assigned by template storage DLL 205 when the template is created and has the form TEMPLEnnn, where nnn may be a decimal number.

Col. 3, lines 1-9 of Buxton teaches storing of a plurality of templates in a memory with a number of indexed locations. The templates are stored in at least one of the indexed memory locations. However, Buxton does not disclose associating each line of command line utility output with a line identifier in the system storage.

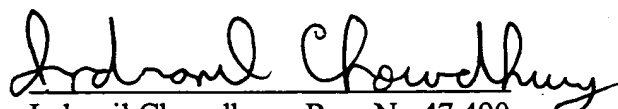
Thus, the rejection of claims 12-14 should be reversed.

IX. CONCLUSION

Applicant requests that each of the final rejections be reversed and that the claims subject to this appeal be allowed to issue.

Respectfully submitted,

Date: June 24, 2003



Indranil Chowdhury, Reg. No.47,490
TROP, PRUNER & HU, P.C.
8554 Katy Freeway, Suite 100
Houston, TX 77024-1805
713/468-8880 [Phone]
713/468-8883 [Facsimile]

APPENDIX OF CLAIMS

The claims on appeal are:

1. A method to provide process command line utility output, comprising:
receiving an identifier;
receiving output from a command line utility; and
storing the command line utility output in a system storage at a location identified by the identifier.
2. The method of claim 1, wherein the act of receiving an identifier comprises receiving an identifier that identifies one or more entries in a system registry database.
3. The method of claim 2, wherein the act of receiving an identifier comprises receiving a root key identifier.
4. The method of claim 3, wherein the act of receiving a root key identifier further comprises receiving a sub-key identifier.
5. The method of claim 2, wherein the system registry comprises an operating system registry database.
6. The method of claim 1, wherein the act of receiving an identifier further comprises receiving a system storage identifier.

7. The method of claim 6, wherein the act of receiving a system storage identifier comprises receiving an identifier indicating a system registry.
8. The method of claim 6, wherein the act of receiving a system storage identifier comprises receiving an identifier indicating shared system memory.
9. The method of claim 8, wherein the act of receiving an identifier indicating shared system memory identifies a system clipboard memory.
10. The method of claim 1, wherein the act of receiving output from a command line utility comprises receiving output directly from the command line output utility.
11. The method of claim 1, wherein the act of receiving output from a command line utility comprises receiving output from the command line output utility through a subsequent command line output routine.
12. The method of claim 1, wherein the act of storing comprises associating each line of command line utility output with a line identifier in the system storage.
13. The method of claim 12, further comprising setting each line identifier to a value corresponding to that lines position in the command line utility output.

14. The method of claim 12, further comprising setting a default value of the received identifier to equal the total number of command utility output lines stored in the system storage.

15. A program storage device, readable by a computer, comprising instructions stored on the program storage device for causing the computer to:

receive an identifier;

receive output from a command line utility; and

store the command line utility output in system storage at a location identified by the identifier.

16. The program storage device of claim 15 wherein the instructions to store comprise instructions to store command line utility output in an operating system registry database.

17. The program storage device of claim 15 wherein the instructions to store comprise instructions to store command line utility output in an operating system maintained volatile memory.

18. The program storage device of claim 15 wherein the instructions to receive output comprise instructions to receive one or more lines of output from the command line utility, and the instructions to store further comprise instructions to store each of said one or more lines of output in the system storage.

19. The program storage device of claim 18 wherein the instructions to store further comprise instructions to associate a unique identifier with each of the one or more lines of output stored in the system storage.

20. The program storage device of claim 18 wherein the instructions to store further comprise instructions to set a value associated with the received identifier in the system storage equal to the number of lines of output stored in the system storage.

21. A computer system, comprising:
a processor; and
a storage device coupled to the processor, the storage device having stored thereon a program having instructions to receive an identifier, receive output from a command line utility, and store the command line utility output in system storage at a location identified by the identifier.

22. The computer system of claim 21, wherein the program comprises a dynamic link library.